



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/823,870	04/14/2004	John Philip MacCormick	226107	6179
41505 7590 08/20/2007 WOODCOCK WASHBURN LLP (MICROSOFT CORPORATION) CIRA CENTRE, 12TH FLOOR 2929 ARCH STREET PHILADELPHIA, PA 19104-2891				
			EXAMINER MYINT, DENNIS Y	
			ART UNIT 2162	PAPER NUMBER
			MAIL DATE 08/20/2007	DELIVERY MODE PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary

Application No.

10/823,870

Applicant(s)

MACCORMICK, JOHN PHILIP

Examiner

Dennis Myint

Art Unit

2162

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) ☒ Responsive to communication(s) filed on 14 June 2007.
- 2a) ☒ This action is **FINAL**. 2b) ☐ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) ☒ Claim(s) 1,7-9,15-17,21,22 and 26 is/are pending in the application.
- 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) ☐ Claim(s) _____ is/are allowed.
- 6) ☒ Claim(s) 1,7-9,15-17,21,22 and 26 is/are rejected.
- 7) ☐ Claim(s) _____ is/are objected to.
- 8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 04/14/2004 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.
- Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
- Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some * c) ☐ None of:
- ☐ Certified copies of the priority documents have been received.
 - ☐ Certified copies of the priority documents have been received in Application No. _____.
 - ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

- ☒ Notice of References Cited (PTO-892)
- ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- ☐ Information Disclosure Statement(s) (PTO/SB/08)
Paper No(s)/Mail Date _____
- ☐ Interview Summary (PTO-413)
Paper No(s)/Mail Date. _____
- ☐ Notice of Informal Patent Application
- ☐ Other: _____

DETAILED ACTION

1. This communication is responsive to Applicant's Amendment, filed on June 14, 2007.
2. Claims 1, 7-9, 15-17, 21, 22, and 26 are pending in this application. Claims 1, 9, 17, and 22 are independent claims. In the Amendment filed on June 14, 2007, Claims 1, 9, 17, and 22 were amended. Claims 2-6, 10-14, and 18-20 were cancelled. **This office action is made final.**
3. In light of the amendments made, rejection of claims 1-8, 25-29, and 31-39 under 35 U.S.C. § 101 in prior office action is hereby withdrawn.

Response to Arguments

4. Applicant's arguments filed on June 14, 2007 have been considered but are moot in view of the new ground(s) of rejection.

Claim Rejections - 35 USC § 103

5. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

Art Unit: 2162

6. Claims 1, 7, 9, 15, 17, 21, 22, and 26 are rejected under 35 U.S.C. 103(a) as being unpatentable over Reiter et al., (hereinafter "Reiter") (U.S. Patent Number 5752243) in view of Bumbulis (hereinafter "Bumbulis") (U.S. Patent Application Publication Number 2003/0204513) and further in view of Burnett (U.S. Patent Application Publication Number 2002/0152226).

As per claim 1, Reiter is directed to a method of changing values of a range of consecutive keys in an original B-tree having a plurality of keys stored therein (Reiter, Figure 8; Column 10 Lines 36-40, i.e., *horizontal splitting*; and Column 9 Lines 32-43, i.e., *Generally, to split a page horizontally, the tree manager moves a subset of the page's node entries to a new adjacent page and stores a new key value and a pointer to the new page in the parent page to the page being split*), comprising:

"excising the range of consecutive keys from the original B-tree" (Reiter, Figure 8; Column 10 Lines 36-40, i.e., ***horizontal splitting***; and Column 9 Lines 32-43, i.e., *Generally, to split a page horizontally, the tree manager **moves a subset of the page's node entries to a new adjacent page and stores a new key value and a pointer to the new page in the parent page to the page being split***; and Reiter, Column 9 Lines 32-43, i.e., ***key value and pointer***), "the original B-tree representing a file system", (Reiter, Column 9 Lines 32-43, i.e., ***key value and pointer***; and Reiter, Column 2 Lines 48-62, i.e., *In a **file system**, for example, **files** and **subdirectories** descend from a main directory. Each successively lower level in the system is a subdirectory that branches from the one above it. A file system contains multiple levels because each directory or subdirectory can contain both files and other directories. In a hierarchical*

Art Unit: 2162

*data model, a user sees data logically organized as a tree, and "units" of **data may themselves be trees or subtrees**. Because B-trees are useful only for flat (i.e., no imbedded structure) homogeneous collections of data, traditional B-tree structures and maintenance algorithms are inadequate for storing hierarchical data models. Even B*-trees, where leaf nodes contain actual units of data, are inadequate because the units of data are not considered to be logically divisible into smaller units)* "wherein renaming an element of the file system requires the changing of the values of the range of consecutive keys" (Reiter, Column 9 Lines 32-43, i.e., *Generally, to split a page horizontally, the tree manager **moves a subset of the page's node entries to a new adjacent page and stores a new key value and a pointer to the new page in the parent page to the page being split***; and Reiter, Column 9 Lines 32-43, i.e., **key value and pointer**) (Note that horizontal splitting would excise a range of consecutive keys from a original B-tree and would convert the original B-tree into a trimmed tree), "the excision of the range of consecutive keys converting the original B-tree into a trimmed tree" (Reiter, Figure 8; Column 10 Lines 36-40, i.e., **horizontal splitting**; and Column 9 Lines 32-43, i.e., *Generally, to split a page horizontally, the tree manager **moves a subset of the page's node entries to a new adjacent page and stores a new key value and a pointer to the new page in the parent page to the page being split***; and Reiter, Column 9 Lines 32-43, i.e., **key value and pointer**);

"balancing the trimmed tree" (Reiter, Column 2 Lines 5-6-15, i.e., *To add a key value to a node that is full, the node is split into two nodes, an additional page is allocated to the B-tree, and one of the two nodes is stored on the new page. The other*

node is stored on the additional page and at the same level of the B-tree as the node that was split. A key value and a pointer to the new node are added to the parent node of the node that was split. When the parent node becomes full, the parent node is also split using the same technique. Splitting can propagate all the way to the root node, creating a new level in the B-tree whenever the root is split; Reiter, Column 3 Lines 35-46, i.e., To split a unit of data vertically, the tree manager first requests allocation of a new page, and then creates and stores a Top of Page (TOP) node as the first node in the new page. The new page is referred to as a step-child page. Next, the tree manager divides the unit of data into first and second subtrees and moves the second subtree to the new page. The second subtree is becomes child nodes to the TOP node. The tree manager stores a pointer node, which contains only a pointer to the step-child page, where the second subtree used to be stored. The tree manager then places a pointer to the pointer node into the subnode table of the split node);

“storing the range of consecutive keys excised from the B-tree to form an extracted tree” (Reiter, Column 9 Lines 32-43, i.e., the tree manager moves a subset of the page’s node entries to a new adjacent page);

*“changing the values of the keys of the extracted tree to form a modified extracted tree” (Reiter, Column 9 Lines 32-43, i.e., Generally, to split a page horizontally, the tree manager moves a subset of the page’s node entries to a new adjacent page and stores **a new key value** and a pointer to the new page in the parent page to the page being split);*

“balancing the (final) B-tree” (Reiter, Column 2 Lines 5-6-15, i.e., *To add a key value to a node that is full, the node is split into two nodes, an additional page is allocated to the B-tree, and one of the two nodes is stored on the new page. The other node is stored on the additional page and at the same level of the B-tree as the node that was split. A key value and a pointer to the new node are added to the parent node of the node that was split. When the parent node becomes full, the parent node is also split using the same technique. Splitting can propagate all the way to the root node, creating a new level in the B-tree whenever the root is split*; Reiter, Column 3 Lines 35-46, i.e., *To split a unit of data vertically, the tree manager first requests allocation of a new page, and then creates and stores a Top of Page (TOP) node as the first node in the new page. The new page is referred to as a step-child page. Next, the tree manager divides the unit of data into first and second subtrees and moves the second subtree to the new page. The second subtree is becomes child nodes to the TOP node. The tree manager stores a pointer node, which contains only a pointer to the step-child page, where the second subtree used to be stored. The tree manager then places a pointer to the pointer node into the subnode table of the split node*);

“wherein the original B-tree represents a hierarchical name space of a file system of a computing device” (Reiter, Column 9 Lines 32-43, i.e., **key value and pointer**; and Reiter, Column 2 Lines 48-62, i.e., *In a **file system**, for example, **files** and **subdirectories** descend from a main directory. Each successively lower level in the system is a subdirectory that branches from the one above it. A file system contains multiple levels because each directory or subdirectory can contain both files and other*

directories. In a hierarchical data model, a user sees data logically organized as a tree, and "units" of data may themselves be trees or subtrees. Because B-trees are useful only for flat (i.e., no imbedded structure) homogeneous collections of data, traditional B-tree structures and maintenance algorithms are inadequate for storing hierarchical data models. Even B-trees, where leaf nodes contain actual units of data, are inadequate because the units of data are not considered to be logically divisible into smaller units).*

Reiter does not explicitly teach the limitation: "inserting the modified extracted tree into the balanced trimmed tree to form a final B-tree", "and the range of consecutive keys belong to a directory of the file system such that the directory is represented as a B-tree, and wherein changing the values of the range of consecutive keys is in connection with the directory being renamed", and "wherein each key in the original B-tree contains a path name for a file or directory of the file system prior to the renaming of the directory".

On the other hand, Bumbulis teaches the limitation: "inserting the modified extracted tree into the balanced trimmed tree to form a final B-tree" (Bumbulis, Paragraph 0115, i.e., *The merge operation is the inverse of the split operation*; Paragraph 0151; Paragraph 0155, i.e., *A merge starts by copying the nodes for the left and right trees into consecutive locations*).

At the time the invention was made, it would have been obvious to a person of ordinary skill in the art to combine the method of Reiter, which performs horizontal splitting and renaming of nodes of a subtree that was split, with the method of Bumbulis,

which merges back the split trees, so that the combined method would excise a range of consecutive keys from an original B-tree, converting the original B-tree into a trimmed tree; store the range of consecutive keys excised from the B-tree to form an extracted tree; change the values of the keys of the extracted tree to form a modified extracted tree; and insert the modified extracted tree into the trimmed tree to form a final B-tree. One would have been motivated to do so in order to *enable a database management system to maintain more compact indexes while providing performance equivalent to existing indexing schemes* (Bumbulis, Paragraph 0022).

Reiter in view of Bumbulis does not explicitly teach the limitations: “and the range of consecutive keys belong to a directory of the file system such that the directory is represented as a B-tree, and wherein changing the values of the range of consecutive keys is in connection with the directory being renamed”, and “wherein each key in the original B-tree contains a path name for a file or directory of the file system prior to the renaming of the directory”.

On the other hand, Burnett teaches the limitations:

“and the range of consecutive keys belong to a directory of the file system such that the directory is represented as a B-tree” (Burnett, Paragraph 0006, i.e., *Situations like this where components higher up in a **directory tree** end up being accessed are actually common*), and “wherein changing the values of the range of consecutive keys is in connection with the directory being renamed”(Burnett Figure 2 and Paragraph 0034, i.e., *FIG. 2 presents the processing steps that occur when an application attempts to access a file and the SCS monitoring component intervenes in the access. The process*

begins with step 200 where the SCS invokes this process and provides as input the path name used to access the underlying file system resource and the type of operation on the resource. The pathname would be one of the many possible variations supported by UNIX such as a full name (e.g. /usr/local/bin/date) or a relative name (../bin/date). The operation describes what type of action is being performed on the resource. Examples of operations could include read, write, change permissions, or rename. The flow proceeds to step 201 where the FID for the target of the access (date) is obtained along with a FID chain for the directory (/usr/local/bin), which is traversed to get at date. The processing of this step is further detailed in FIG. 4 of the present invention. Step 202 searches the FID to DN/SCC mapping database to locate an SCC definition. Step 202 finds the effective SCC and DN from the FID or FID chain. The search of the mapping database is described in greater detail in FIG. 3 of the present invention. If no SCC is found in step 203, the process proceeds to step 204 where the SCC definition is defaulted to the category "unclassified" and the DN is set to the path used in the file system resource access. Step 204 would then proceed to back to the main processing path indicated by step 205. If in step 203 a SCC was found and returned in the mapping database search, the flow proceeds directly to step 205 where the accounting data maintained by the SCS is updated. An update could include incremented the count of accesses for the given classification and perhaps adding a record to the monitoring log consisting of the DN corresponding to the access and the classification defined for the DN. The process then proceeds to step 206 where the desired operation is checked to determine if the operation could affect the file system name space. An operation such

as a **rename on a directory** could have such an effect and potentially render a cached **FID chain** in the FID chain cache (FCC) as stale. For example, if /usr/local where renamed to /usr/loc, then any FCC entry containing a FID for local would be stale and would require invalidation. The invalidation logic is detail in FIG. 6. If in step 206, the operation requires an FCC flush, step 207 is invoked to flush the entries related to path. Step 207 then proceed back to the main logic flow which ends with step 208. If in step 206, the operation does not affect the file system name space, then logic proceeds directly to stop 208 where the process of FIG. 2 ends. Particularly note that FID chain is invalidated and regenerated when a directory is renamed. FIDs in the FID chain are equivalent to consecutive keys of the claimed invention), and

“wherein each key in the original B-tree” (Burnett, Paragraph, i.e., *Situations like this where components higher up in a **directory tree** end up being accessed are actually common*) and “contains a path name for a file or directory of the file system prior to the renaming of the directory” (Burnett, Abstract, *The techniques of the present invention involve constructing and caching a chain of FIDs that represent the directory path to a system resource. Typically a lookupname() service returns a **handle to the object and a handle to the object's owning directory**. The owning directory handle can then be used to obtain a file identifier (FID) for the owning directory. This FID constitutes the first FID in the chain and will also act as the lookup key in the cache. The process next finds the directory's parent. A FID is obtained with the parent handle. This FID is added to the chain. The process repeats until the root of the system's file tree is reached. This result is a chain or array of FIDs representing the full path name of the*

Art Unit: 2162

directory containing the accessed object. The chain begins with the directory FID closest to the accessed object and ends with the FID representing the root of the file tree).

At the time the invention was made, it would have been obvious to a person of ordinary skill in the art to modify the method of Reiter in view of Bumbulis to add the features of representing a directory as a B-tree, changing the values of the range of consecutive keys in connection with the directory being renamed, and making each key in the original B-tree contain a path name for a file for directory of the file system prior to the renaming of the directory, as taught by Burnett, so that the resultant method would also comprise representing a directory as a B-tree, changing the values of the range of consecutive keys in connection with the directory being renamed, and making each key in the original B-tree contain a path name for a file for directory of the file system prior to the renaming of the directory. One would have been motivated to do so in order to improve the lookup operations on a file system resource name (Burnett, paragraph 0009).

As per claim 7, Bumbulis in view of Reiter and further in view of Burnett teaches the limitation:

“ wherein the step of changing includes changing a prefix field of a root node”
(Bumbulis, Figures 7E and 7E and Paragraph 0198, i.e., *Fig. 7D illustrates a ptree 730 after insertion of a node y having a key value key (y) = prefix (c) 0 into the ptree 710 of Fig 7A*) “of the extracted tree” (Reiter, Figure 8; Column 10 Lines 36-40, i.e., *horizontal*

splitting; and Column 9 Lines 32-43, i.e., Generally, to split a page horizontally, the tree manager moves a subset of the page's node entries to a new adjacent page and stores a new key value and a pointer to the new page in the parent page to the page being split).

Claim 9 are essentially the same as claim 1 except that it set forth the claimed invention as a computer-readable medium rather than a method and rejected for the same reasons as applied hereinabove.

Claim 15 are essentially the same as claim 7 except that it set forth the claimed invention as a computer-readable medium rather than a method and rejected for the same reasons as applied hereinabove.

As per claim 17, Reiter in view of Bumbulis and further in view of Burnett is directed to a method of modifying a B-tree, wherein the B-tree represents a file system of a computing device, wherein renaming an element of the file system requires the changing of the values of the range of consecutive keys, each key in the B-tree contains a pathname for a file for directory of the file system (Reiter, Column 9 Lines 32-43, i.e., *key value and pointer*; and Rao, Column 11 Lines 52-67, i.e., *by means of a pathname of the directory*; Burnett, Paragraph 0006, i.e., *Situations like this where components higher up in a **directory tree** end up being accessed are actually common*; and Burnett, Abstract, *The techniques of the present invention involve constructing and caching a*

Art Unit: 2162

*chain of FIDs that represent the directory path to a system resource. Typically a lookupname() service returns **a handle to the object and a handle to the object's owning directory**. The owning directory handle can then be used to obtain a file identifier (FID) for the owning directory. This FID constitutes the first FID in the chain and will also act as the lookup key in the cache. The process next finds the directory's parent. A FID is obtained with the parent handle. This FID is added to the chain. The process repeats until the root of the system's file tree is reached. This result is a chain or array of FIDs representing the full path name of the directory containing the accessed object. The chain begins with the directory FID closest to the accessed object and ends with the FID representing the root of the file tree)* and teaches the limitations:

“excising keys of a directory of the file system being renamed from the B-tree, the excision of the keys of the directory converting the B-tree into a trimmed tree” (Reiter, Figure 8; Column 10 Lines 36-40, i.e., *horizontal splitting*; and Column 9 Lines 32-43, i.e., *Generally, to split a page horizontally, the tree manager moves a subset of the page's node entries to a new adjacent page and stores a new key value and a pointer to the new page in the parent page to the page being split*; Note that horizontal splitting would excise a range of consecutive keys from a original B-tree and would convert the original B-tree into a trimmed tree; and Burnett Figure 2 and Paragraph 0034, i.e., *FIG. 2 presents the processing steps that occur when an application attempts to access a file and the SCS monitoring component intervenes in the access. The process begins with step 200 where the SCS invokes this process and provides as input the path name used to access the underlying file system resource and the type of operation on the*

resource. The pathname would be one of the many possible variations supported by UNIX such as a full name (e.g. /usr/local/bin/date) or a relative name (../bin/date). The operation describes what type of action is being performed on the resource. Examples of operations could include read, write, change permissions, or rename. The flow proceeds to step 201 where the FID for the target of the access (date) is obtained along with a FID chain for the directory (/usr/local/bin), which is traversed to get at date. The processing of this step is further detailed in FIG. 4 of the present invention. Step 202 searches the FID to DN/SCC mapping database to locate an SCC definition. Step 202 finds the effective SCC and DN from the FID or FID chain. The search of the mapping database is described in greater detail in FIG. 3 of the present invention. If no SCC is found in step 203, the process proceeds to step 204 where the SCC definition is defaulted to the category "unclassified" and the DN is set to the path used in the file system resource access. Step 204 would then proceed to back to the main processing path indicated by step 205. If in step 203 a SCC was found and returned in the mapping database search, the flow proceeds directly to step 205 where the accounting data maintained by the SCS is updated. An update could include incremented the count of accesses for the given classification and perhaps adding a record to the monitoring log consisting of the DN corresponding to the access and the classification defined for the DN. The process then proceeds to step 206 where the desired operation is checked to determine if the operation could affect the file system name space. An operation such as a **rename on a directory** could have such an effect and potentially render a cached **FID chain** in the FID chain cache (FCC) as stale. For example, if /usr/local where

renamed to /usr/loc, then any FCC entry containing a FID for local would be stale and would require invalidation. The invalidation logic is detail in FIG. 6. If in step 206, the operation requires an FCC flush, step 207 is invoked to flush the entries related to path. Step 207 then proceed back to the main logic flow which ends with step 208. If in step 206, the operation does not affect the file system name space, then logic proceeds directly to stop 208 where the process of FIG. 2 ends. Particularly note that FID chain is invalidated and regenerated when a directory is renamed. FIDs in the FID chain are equivalent to consecutive keys of the claimed invention);

“balancing the trimmed tree” (Reiter, Column 2 Lines 5-6-15, i.e., To add a key value to a node that is full, the node is split into two nodes, an additional page is allocated to the B-tree, and one of the two nodes is stored on the new page. The other node is stored on the additional page and at the same level of the B-tree as the node that was split. A key value and a pointer to the new node are added to the parent node of the node that was split. When the parent node becomes full, the parent node is also split using the same technique. Splitting can propagate all the way to the root node, creating a new level in the B-tree whenever the root is split; Reiter, Column 3 Lines 35-46, i.e., To split a unit of data vertically, the tree manager first requests allocation of a new page, and then creates and stores a Top of Page (TOP) node as the first node in the new page. The new page is referred to as a step-child page. Next, the tree manager divides the unit of data into first and second subtrees and moves the second subtree to the new page. The second subtree is becomes child nodes to the TOP node. The tree manager stores a pointer node, which contains only a pointer to the step-child page,

where the second subtree used to be stored. The tree manager then places a pointer to the pointer node into the subnode table of the split node);

"storing the keys of the directory excised from the B-tree in an extracted tree" (Reiter, Column 9 Lines 32-43, i.e., the tree manager moves a subset of the page's node entries to a new adjacent page);

"changing the values of the keys of the extracted tree to reflect a new name of the directory" (Reiter, Column 9 Lines 32-43, i.e., Generally, to split a page horizontally, the tree manager moves a subset of the page's node entries to a new adjacent page and stores a new key value and a pointer to the new page in the parent page to the page being split; and Burnett Figure 2 and Paragraph 0034, i.e., FIG. 2 presents the processing steps that occur when an application attempts to access a file and the SCS monitoring component intervenes in the access. The process begins with step 200 where the SCS invokes this process and provides as input the path name used to access the underlying file system resource and the type of operation on the resource. The pathname would be one of the many possible variations supported by UNIX such as a full name (e.g. /usr/local/bin/date) or a relative name (../bin/date). The operation describes what type of action is being performed on the resource. Examples of operations could include read, write, change permissions, or rename. The flow proceeds to step 201 where the FID for the target of the access (date) is obtained along with a FID chain for the directory (/usr/local/bin), which is traversed to get at date. The processing of this step is further detailed in FIG. 4 of the present invention. Step 202 searches the FID to DN/SCC mapping database to locate an SCC definition. Step 202

*finds the effective SCC and DN from the FID or FID chain. The search of the mapping database is described in greater detail in FIG. 3 of the present invention. If no SCC is found in step 203, the process proceeds to step 204 where the SCC definition is defaulted to the category "unclassified" and the DN is set to the path used in the file system resource access. Step 204 would then proceed to back to the main processing path indicated by step 205. If in step 203 a SCC was found and returned in the mapping database search, the flow proceeds directly to step 205 where the accounting data maintained by the SCS is updated. An update could include incremented the count of accesses for the given classification and perhaps adding a record to the monitoring log consisting of the DN corresponding to the access and the classification defined for the DN. The process then proceeds to step 206 where the desired operation is checked to determine if the operation could affect the file system name space. An operation such as a **rename on a directory** could have such an effect and potentially render a cached **FID chain** in the FID chain cache (FCC) as stale. For example, if /usr/local where renamed to /usr/loc, then any FCC entry containing a FID for local would be stale and would require invalidation. The invalidation logic is detail in FIG. 6. If in step 206, the operation requires an FCC flush, step 207 is invoked to flush the entries related to path. Step 207 then proceed back to the main logic flow which ends with step 208. If in step 206, the operation does not affect the file system name space, then logic proceeds directly to stop 208 where the process of FIG. 2 ends. Particularly note that FID chain is invalidated and regenerated when a directory is renamed. FIDs in the FID chain are equivalent to consecutive keys of the claimed invention); and*

“inserting the extracted tree with changed values of the keys into the trimmed tree to form a final B-tree” (Bumbulis, Paragraph 0115, i.e., *The merge operation is the inverse of the split operation*; Paragraph 0151; Paragraph 0155, i.e., *A merge starts by copying the nodes for the left and right trees into consecutive locations*); and

“balancing the final B-tree” (Reiter, Column 2 Lines 5-6-15, i.e., *To add a key value to a node that is full, the node is split into two nodes, an additional page is allocated to the B-tree, and one of the two nodes is stored on the new page. The other node is stored on the additional page and at the same level of the B-tree as the node that was split. A key value and a pointer to the new node are added to the parent node of the node that was split. When the parent node becomes full, the parent node is also split using the same technique. Splitting can propagate all the way to the root node, creating a new level in the B-tree whenever the root is split*; Reiter, Column 3 Lines 35-46, i.e., *To split a unit of data vertically, the tree manager first requests allocation of a new page, and then creates and stores a Top of Page (TOP) node as the first node in the new page. The new page is referred to as a step-child page. Next, the tree manager divides the unit of data into first and second subtrees and moves the second subtree to the new page. The second subtree is becomes child nodes to the TOP node. The tree manager stores a pointer node, which contains only a pointer to the step-child page, where the second subtree used to be stored. The tree manager then places a pointer to the pointer node into the subnode table of the split node*).

As per claim 21, Bumbulis in view of Reiter and further in view of Burnett teaches the limitation:

“ wherein the step of changing includes changing a prefix field of a root node”
(Bumbulis, Figures 7E and 7E and Paragraph 0198, i.e., *Fig. 7D illustrates a ptree 730 after insertion of a node y having a key value key (y) = prefix (c) 0 into the ptree 710 of Fig 7A*) “of the extracted tree” (Reiter, Figure 8; Column 10 Lines 36-40, i.e., *horizontal splitting*; and Column 9 Lines 32-43, i.e., *Generally, to split a page horizontally, the tree manager moves a subset of the page’s node entries to a new adjacent page and stores a new key value and a pointer to the new page in the parent page to the page being split*).

Claim 22 are essentially the same as claim 17 except that it set forth the claimed invention as a computer-readable medium rather than a method and rejected for the same reasons as applied hereinabove.

Claim 26 are essentially the same as claim 21 except that it set forth the claimed invention as a computer-readable medium rather than a method and rejected for the same reasons as applied hereinabove.

7. Claims 8 and 16 are rejected under 35 U.S.C. 103(a) as being unpatentable over Reiter in view of Bumbulis and further in view of Burnett and further in view of Cheng et

al., (hereinafter "Cheng") (U.S. Patent Number 5204958) and further in view of Beyer et al., (hereinafter "Beyer") (U.S. Patent Application Publication Number 2006/0173927).

As per claim 8, Reiter in view of Bumbulis and further in view of Burnett teaches the limitation: "wherein the step of inserting the modified extracted tree into the trimmed tree" (Bumbulis, Paragraph 0115, i.e., *The merge operation is the inverse of the split operation*; Paragraph 0151; Paragraph 0155, i.e., *A merge starts by copying the nodes for the left and right trees into consecutive locations*).

Reiter in view of Bumbulis and further in view of Burnett does not explicitly teach the limitation: "involves a strict insertion".

Beyer teaches the limitation: "strict insertion" (Figure 1 and Paragraph 0022-0024, and particularly Paragraph 0017, i.e., *still maintain these properties*).

At the time the invention was made, it would have been obvious to a person ordinary skill in the art to modify the method of Reiter in view of Bumbulis and further in view of Burnett to add the feature of using a strict insertion by way of maintaining key values (i.e. ID values and relationships among nodes), as taught by Beyer, to the method of Reiter in view of Bumbulis and further in view of Burnett so that the resultant method would comprise the step of inserting the modified extracted tree into the trimmed tree involves a strict insertion. One would have been motivated to do so in order to *to maintain/retain the order and relationships between the parent, child, sibling nodes* (Beyer, Paragraph 0015).

Art Unit: 2162

Claim 16 are essentially the same as claim 8 except that it set forth the claimed invention as a computer-readable medium rather than a method and rejected for the same reasons as applied hereinabove.

Conclusion

8. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).


A shortened statutory period for reply to this final action is set to expire **THREE MONTHS** from the mailing date of this action. In the event a first reply is filed within **TWO MONTHS** of the mailing date of this final action and the advisory action is not mailed until after the end of the **THREE-MONTH** shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than **SIX MONTHS** from the date of this final action.


Contact Information

9. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Dennis Myint whose telephone number is (571) 272-5629. The examiner can normally be reached on 8:30AM-5:30PM Monday-Friday.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, John Breene can be reached on (571) 272-4107. The fax phone number for the organization where this application or proceeding is assigned is 571-273-5629.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).


SHAHID ALAM
PRIMARY EXAMINER


Dennis Myint
Examiner
AU-2162